Evaluation of Large Integer Multiplication Methods on Hardware

Evaluation of Large Integer Multiplication Methods on Hardware

Ciara Rafferty, *Member, IEEE*, Máire O'Neill, *Senior Member, IEEE*, Neil Hanley

Abstract—Multipliers requiring large bit lengths have a major impact on the performance of many applications, such as cryptography, digital signal processing (DSP) and image processing. Novel, optimised designs of large integer multiplication are needed as previous approaches, such as schoolbook multiplication, may not be as feasible due to the large parameter sizes. Parameter bit lengths of up to millions of bits are required for use in cryptography, such as in lattice-based and fully homomorphic encryption (FHE) schemes. This paper presents a comparison of hardware architectures for large integer multiplication. Several multiplication methods and combinations thereof are analysed for suitability in hardware designs, targeting the FPGA platform. In particular, the first hardware architecture combining Karatsuba and Comba multiplication is proposed. Moreover, a hardware complexity analysis is conducted to give results independent of any particular FPGA platform. It is shown that hardware designs of combination multipliers, at a cost of additional hardware resource usage, can offer lower latency compared to individual multiplier designs. Indeed, the proposed novel combination hardware design of the Karatsuba-Comba multiplier offers lowest latency for integers greater than 512 bits. For large multiplicands, greater than 16384 bits, the hardware complexity analysis indicates that the NTT-Karatsuba-Schoolbook combination is most suitable.

Index Terms—Large integer multiplication, FPGA, hardware complexity, fully homomorphic encryption

1 Introduction

Large integer multiplication is a key component and one of the bottlenecks within many applications, such as cryptographic schemes. More specifically, important and widely used public key cryptosystems, such as RSA and elliptic curve cryptography (ECC), require multiplication. Such public key cryptosystems are used along with symmetric cryptosystems within the Transport Layer Security (TLS) protocol, to enable secure online communications. Thus, there is a demand for efficient, optimised implementations and, to this end, optimised hardware designs are commonly used to improve the performance of multipliers.

To demonstrate the importance of suitable hardware multipliers for large integer multiplication, a case study on a specific branch of cryptography called fully homomorphic encryption (FHE) is detailed. FHE, introduced in 2009 [1], is a novel method of encryption, which allows computation on encrypted data. Thus, this property of FHE can potentially advance areas such as secure cloud computation and secure multi-party computation [2], [3]. However, existing FHE schemes are currently highly unpractical due to large parameter sizes and highly computationally intensive algorithms amongst other issues. Therefore improvements in the practicality of FHE schemes will have a large impact on both cloud security and the usage of cloud services. There has been recent research into theoretical optimisations and both software and hardware designs of FHE schemes to improve their practicality; hardware designs have been shown to greatly increase performance [4]–[13]. Indeed, several researchers studying the hardware design for FHE have focused on the multiplication component to enhance the practicality of FHE schemes [11], [14]–[16]. This highlights the importance of selecting the most suitable multiplication method for use with large operands. Previous hardware designs have mostly chosen multiplication using the number theoretic transform (NTT) for the large integer multiplication required in FHE schemes since this method is known generally to be suitable for large integer multiplication; however, there has been little research into the use of alternative large integer multiplication methods or indeed into multipliers of the operand sizes required for FHE schemes.

Previous research has investigated and compared multipliers in hardware and more particularly for use in public key cryptography [17]–[22]. Modular multiplication has been investigated and Montgomery multipliers have been optimised for use in public key cryptography [17], [19]–[21]. An analysis of the hardware complexity of several multipliers for modular multiplication and modular exponentiation for use in public key cryptography has shown Karatsuba outperforms traditional schoolbook multiplication for operands greater than or equal to 32 bits [17]. Fast Fourier transform (FFT) multiplication is also shown to perform better than classical schoolbook multiplication for larger operands [17]. In Section 2, several common multiplication methods are detailed and the previous research into hardware designs is also discussed for each technique.

However, larger integer multiplication, such as those required in new public key encryption schemes like FHE, has not been previously investigated. While some previous research looks at multiplications for specific applications, to the best of the authors' knowledge, there is no prior research that analyses and compares hardware designs of various multiplication algorithms for very large integers,

C. Rafferty, M. O'Neill and N. Hanley are with the Centre for Secure Information Technologies (CSIT), Queen's University Belfast, Northern Ireland (e-mail: {c.m.rafferty, maire.oneill, n.hanley}@qub.ac.uk)

greater than 4096 bits. The authors have carried out previous research on hardware designs for optimised multiplication for use specifically in FHE schemes [13], [16], which offer targeted designs for one particular FHE scheme. In this research, hardware multiplier designs are considered for a large range of operand sizes and in a wider context, for a generic large integer multiplication. Moreover, to the best of the authors' knowledge, suitable multiplication methods for uneven operands, such as those required in the integer-based FHE encryption scheme [23], have also not previously been investigated. Also, hardware designs of combination multiplication methods have also not yet been considered. It is thus posed in this research that if hardware designs of combined multiplication methods could improve performance compared to hardware designs of individual multiplication methods.

More specifically, the novel contributions presented in this research are:

1) A comprehensive evaluation of very large integer multiplication methods;
2) Novel combinations of common multiplication methods and respective hardware designs are proposed;
3) The first study of multiplication with uneven operands;
4) A hardware complexity analysis is presented for all of the proposed multiplication methods and recommendations are given from both the theoretical complexity analysis and hardware results.

The structure of the paper is as follows: firstly, a background of the most popular multiplication methods is presented. Secondly, hardware designs of a selection of the large integer multiplication methods are presented; these are particularly suited or targeted to the application of FHE. Following this, hardware designs of combinations of these multipliers are presented. In Section 5, the hardware complexity of the proposed multipliers is theoretically calculated and recommendations are given on the most suitable multiplication methods for large integer multiplication. All of the proposed multipliers are implemented on a Virtex-7 FPGA and performance results are discussed. The FPGA platform is suitable for numerous applications including cryptography, since such platforms are highly flexible, cost-effective and reprogrammable. Finally, a discussion on suitable multiplication methods for uneven operands is included, which is applicable to integer-based FHE, and conclusions with overall recommendations are given.

2 Multiplication Methods

The following subsections outline the most commonly used multiplication methods for traditional integer multiplication:

2.1 Traditional Schoolbook Multiplication

Schoolbook multiplication is the traditional method that uses shift and addition operations to multiply two inputs. Algorithm 1 outlines the Schoolbook multiplication method.

Algorithm 1: Traditional Schoolbook Multiplication

Input: n-bit integers a and b
Output: $z = a \times b$
1: **for** i in 0 to $n-1$ **do**
2: **if** $b_i = 1$ **then**
3: $z = (a \times 2^i) + z$;
4: **end if**
5: **end for**
return z

2.2 Comba Multiplication Scheduling

Comba multiplication [24] is an optimised method of scheduling the partial products in multiplication, and it differs from the traditional schoolbook method in the ordering of the partial products for accumulation. Algorithm 2 describes Comba multiplication. Although this method is theoretically no faster than the schoolbook method, it is particularly suitable for optimised calculation of partial products. Comba multiplication has previously been considered for modular multiplication on resource restricted devices such as smart cards [25]. A hardware design of the Comba multiplication technique has been previously shown to be suitable for cryptographic purposes [26]. The use of a Comba scheduling algorithm targeting the DSP slices available on a FPGA device reduces the number of read and write operations by managing the partial products in large integer multiplication.

Algorithm 2: Comba Multiplication

Input: n-bit integers a and b
Output: $z = a \times b$
1: **for** i in 0 to $(2n-2)$ **do**
2: **if** $n < i$ **then**
3: $pp_i = \sum_{k=0}^{i-1}(a_k \times b_{i-k})$
4: **else**
5: $pp_i = \sum_{k=0}^{n-1}(a_k \times b_{i-k})$
6: **end if**
7: **end for**
8: $z = \sum_{i=0}^{2n-2}(pp_i << 2^i)$
return z

Another advantage of the Comba scheduling method is that the number of required DSP48E1 blocks available on the target device, in this case a Xilinx Virtex-7 XC7VX980T FPGA, scales linearly with the bit length. This is advantageous when designing larger multipliers, such as those required in FHE schemes. However, the inherent architecture of this algorithm inhibits a pipelined design and thus the need for the encryption of multiple values may be better addressed with alternative methods.

2.3 Karatsuba Multiplication

Karatsuba multiplication [27] was one of the first multiplication techniques proposed to improve on the schoolbook multiplication method, which consists of a series of shifts and additions. The Karatsuba method involves dividing each large integer multiplicand into two smaller integers,

one of which is multiplied by a base. Algorithm 3 details Karatsuba multiplication.

Algorithm 3: Karatsuba Multiplication

Input: n-bit integers a and b, where $a = a_1 \times 2^l + a_0$ and $b = b_1 \times 2^l + b_0$
Output: $z = a \times b$
1: $AB_0 = a_0 b_0$
2: $AB_1 = a_1 b_1$
3: $ADD_A = a_1 + a_0$
4: $ADD_B = b_1 + b_0$
5: $AB_2 = ADD_A \times ADD_B$
6: $MID_0 = AB_2 - AB_0 - AB_1$
7: $z = AB_0 + MID_0 \times 2^l + AB_1 \times 2^{2l}$
return z

In general, if we take two n-bit integers, a and b, to be multiplied, and we take a base, for example $2^{\lceil n/2 \rceil}$, then a and b are defined in Equations 1 and 2 respectively.

$$a = a_1 \times 2^{\lceil n/2 \rceil} + a_0 \tag{1}$$

$$y = y_1 \times 2^{\lceil m/2 \rceil} + y_0 \tag{2}$$

The Karatsuba multiplication method takes advantage of Equation 3. As can be seen in Equation 3, three different multiplications of roughly $\lceil n/2 \rceil$-bit multiplicand sizes are necessary, as well as several subtractions and additions, which are generally of minimal hardware cost in comparison to multiplication operations.

$$\begin{aligned} a \times b = {} & (a_1 \times b_1) \times 2^{2 \times \lceil n/2 \rceil} \\ & + \{(a_1 + a_0) \times (b_1 + b_0) - a_0 \times b_0 - a_1 \times b_1\} \times 2^{\lceil n/2 \rceil} \\ & + a_0 \times b_0 \end{aligned} \tag{3}$$

Thus, Karatsuba is a fast multiplication method, and improves on the schoolbook multiplication. However, several intermediate values must be stored and therefore this method incurs some additional hardware storage cost and also more control logic is required.

There has been a significant amount of research carried out on the Karatsuba algorithm and several optimised hardware implementations have been proposed; for example, a hardware design of a Montgomery multiplier which includes a Karatsuba algorithm has previously been presented [28]. A recursive Karatsuba algorithm is used, breaking multiplications down to 32-bits on a Xilinx Virtex-6 FPGA; this design offers a speedup factor of up to 190 compared to software but consumes a large amount of resources. Another Karatsuba design has targeted the Xilinx Virtex-5 FPGA platform and uses minimal resources (only one DSP48E block) by employing a recursive design [18]. Surveys on earlier research into the hardware designs of Karatsuba and other multiplication methods also exist [17], [29].

There have also been several algorithmic optimisations and extensions to the Karatsuba algorithm. A Karatsuba-like multiplication algorithm with reduced multiplication requirements for multiplying five, six and seven term polynomials has been proposed [30]. A comparison is given of this proposed algorithm, which uses five term polynomials and requires 14 multiplications, with alternative Toom-Cook and FFT algorithms implemented in software. According to this research, the FFT algorithm is the most suitable for large multiplications.

Karatsuba has been shown to be useful for cryptographic purposes [31]; an extended Karatsuba algorithm adapted to be more suitable for hardware implementation for use in computing bilinear pairings has been presented [31]. For a 256-bit multiplication, 14 64-bit products are required, compared to 25 for schoolbook multiplication.

2.4 Toom-Cook Multiplication

Toom-Cook multiplication is essentially an extension of Karatsuba multiplication; this technique was proposed by Toom [32] and extended by Cook [33]. The main difference between the Karatsuba and Toom-Cook multiplication methods is that in the latter, the multiplicands are broken up into several smaller integers, for example three or more integers, whereas Karatsuba divides multiplicands into two integers. Toom-Cook algorithms are used in the GMP library for mid-sized multiplications [34]. The Karatsuba hardware design could be adapted to carry out Toom-Cook multiplication; however the hardware design of a Toom-Cook multiplication requires several more intermediate values, and thus occupies more area resources on hardware devices. For this reason, this multiplication technique is not addressed further in this comparison study.

2.5 Montgomery Modular Multiplication

The discussion of multiplication methods for cryptography would not be complete without the mention of Montgomery modular multiplication [35]. This method of multiplication incorporates a modular reduction and therefore is suitable for many cryptosystems, for example those working in finite fields. Equation 4 gives the calculation carried out by a modular multiplication, with a modulus p and two integers a and b less than p.

$$c \equiv a \cdot b \, mod \, p \tag{4}$$

There has been a lot of research looking into hardware architectures for fast Montgomery reduction and multiplication [20]. Montgomery modular multiplication however requires pre- and post-processing costs to convert values to and from the Montgomery domain. Therefore this method is highly suitable for exponentiations, such as those required in cryptosystems such as RSA. The integer-based FHE scheme does not require exponentiations and the aim of this research is speed, so the conversions to and from the Montgomery domain are considered expensive. For this reason, Montgomery modular reduction is not considered in this research.

2.6 Number Theoretic Transforms for Multiplication

NTT multiplication is arguably the most popular method for large integer multiplication. Almost all of the previous hardware architectures for FHE schemes incorporate an NTT multiplier for large integer multiplication. Algorithm 4 outlines NTT multiplication.

Algorithm 4: Large integer multiplication using NTT [36], [37]

Input: n-bit integers a and b, base bit length l, NTT-point k
Output: $z = a \times b$

1: a and b are n-bit integers. Zero pad a and b to $2n$ bits respectively;
2: The padded a and b are then arranged into k-element arrays respectively, where each element is of length l-bits;
3: **for** i in 0 to $k-1$ **do**
4: $A_i \leftarrow NTT(a_i)$;
5: $B_i \leftarrow NTT(b_i)$;
6: **end for**
7: **for** i in 0 to $k-1$ **do**
8: $Z_i \leftarrow A_i \cdot B_i$;
9: **end for**
10: **for** i in 0 to $k-1$ **do**
11: $z_i \leftarrow INTT(Z_i)$;
12: **end for**
13: **for** i in 0 to $k-1$ **do**
14: $z = \sum_{i=0}^{k-1} (z_i \ll (i \cdot l))$, where $\ll$ is the left shift operation;
15: **end for**
return z

The number theoretic transform (NTT) is a specific case of the FFT over a finite field Z. The NTT is chosen for large integer multiplication within FHE schemes rather than the traditional FFT using complex numbers because it allows exact computations on fixed point numbers. Thus, it is very suitable for cryptographic applications as cryptographic schemes usually require exact computations. Often in the FHE literature, the NTT is referred to more generally as the FFT. However, almost all hardware and software designs of FHE schemes that use FFT are, more specifically, using the NTT. The library proposed by [38] gives the only existing FHE software or hardware design which uses the FFT with complex numbers rather than the NTT with roots of unity. The use of the NTT is particularly appropriate for hardware designs of FHE schemes as a highly suitable modulus can be chosen, which offers fast modular reduction due to the modulus structure.

Modular reduction is required in all FHE schemes and also in NTT multiplications. There are several methods for the modular reduction operation, such as Barrett reduction and also Montgomery modular reduction. However, if the modulus can be specifically chosen, such as within NTT multiplication, certain moduli values lend themselves to efficient reduction techniques. Previous research has also proposed the use of a Solinas prime modulus [37]. Further examples of special number structures for optimised modular reduction include Mersenne and Fermat numbers [39].

For fast polynomial multiplication designs for lattice-based cryptography, the largest known Fermat prime $p = 65537 = 2^{16}+1$ has been used [7]. Alternatively, researchers have used larger prime moduli with a low Hamming weight, such as the modulus $p = 1049089 = 2^{20} + 2^9 + 1$ which has a Hamming weight of 3, [7], [40]. A modular multiplier architecture incorporating a Fermat number modulus is also proposed by [41] for use in a lattice-based primitive. Diminished one representation [42] has been shown to be suitable for moduli of the form $2^n + 1$ and could be considered as an optimisation.

Several hardware designs of FFT multiplication of large integers have been proposed [17], [43]–[45]. Some research has been conducted into the design of FFT multipliers for cryptographic purposes and more specifically for use within lattice-based cryptography [40], [41], [46]. It can be seen from the previously mentioned hardware NTT multiplier architectures, that the hardware design of an efficient NTT multiplier involves several design and optimisation decisions and trade-offs.

3 Hardware Designs for Multipliers

3.1 Direct Multiplication

Direct multiplication is the optimised multiplication method that can be employed in a single clock cycle using a basic VHDL multiplication operation within the Xilinx ISE design suite. In this research, ISE Design Suite 14.1 is used, and a direct multiplication is arbitrarily used as a base standard for multiplication to indicate the performance of the following proposed hardware multiplier designs.

3.2 Comba Multiplication

Comba multiplication [24] is a method of ordering and scheduling partial products. Previously, Güneysu optimised the Comba multiplication by maximising the hardware resource usage and minimising the required number of read and write operations for use in elliptic curve cryptography [26]. A multiplication of two n- word numbers produces $2n-1$ partial products, given any word of arbitrary bit length. Figure 1 outlines the proposed hardware architecture of the Comba multiplier in this research targeting the Xilinx Virtex-7 platform and in particular the available DSP slices, as previously proposed for use in the design of an encryption step for FHE schemes [16], [47]. The abundant Xilinx DSP48E1 slices available on Virtex-7 FPGAs are specifically optimised for DSP operations. Each slice offers a 48-bit accumulation unit and an 18×25-bit signed multiplication block [48]. These DSP slices can run at frequencies of up to 741 MHz [49].

For the multiplication of $a \times b$, where $b \leq a$, the multiplicands are each divided into s blocks, where for example $s = \left\lceil \frac{bit\_length(a)}{w} \right\rceil$. This can be seen in Figure 1. Each block multiplicand is then of the size w bits, which is the size of the next power of two greater than or equal to the bit length of the b operand. Powers of two are used to maximise the efficiency of operations such as shifting. Both multiplicands are stored in 16-bit blocks in registers. Although 18×25-bit signed multiplications are possible within each DSP slice, a

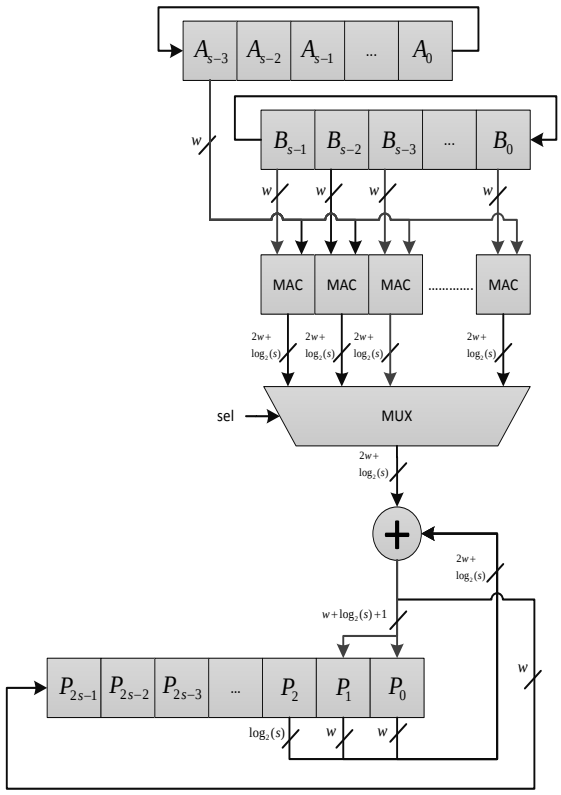

Fig. 1. Comba multiplier architecture [16], [47]

16-bit multiplication input is chosen to ensure efficient computation on the FPGA platform. These blocks are shifted in opposing directions and input into the multiply-accumulate (MAC) blocks in the DSP slices. The partial multiplications are accumulated in each of the MAC blocks.

3.3 Proposed NTT Hardware Multiplier Architecture for Comparison

In this section, a 'simple' NTT module is presented, as illustrated in Figure 2. The scope of this research is not to produce a novel, optimal NTT or FFT architecture. Indeed, there has been a plethora of research in this area. The NTT module discussed here is used for comparison purposes with other multiplication architectures and could be further improved.

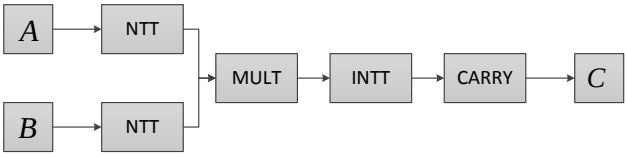

Fig. 2. Architecture of a basic NTT multiplier

A radix-2 decimation in time (DIT) approach is used in this NTT module. At each stage the block of butterfly units is re-used and the addresses managed in order to minimise hardware resource usage. Moreover, in this design, the NTT module is optimised for re-use since both an NTT module and an inverse NTT (INTT) module are required; this minimises resource usage. This can be seen in Figure 3.

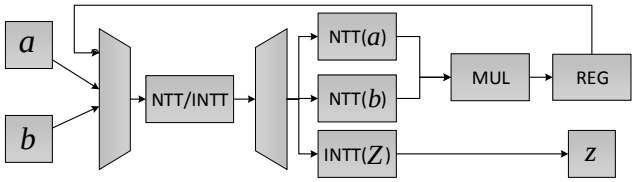

Fig. 3. Architecture of NTT multiplier with optimised NTT module reuse

The advantage of NTT multiplication can be particularly noticed when several multiplications are required, rather than a single multiplication. This is because NTT designs can be pipelined (and staged) so that many operations can be carried out in parallel. Thus, for a single multiplication, NTT multiplication may prove too costly, in terms of both hardware resource usage and latency. However, if multiple multiplications are required, the latency will be reduced though the use of a pipelined design. The NTT design referred to in the rest of this research is a design using parallel butterflies to minimise latency.

4 Proposed Multiplication Architecture Combinations

In this section, hardware architectures for combinations of the previously detailed multiplications are proposed. The aim of these combinations is to increase the speed of the multiplication for use within FHE schemes. More generally, this research aims to show that a hardware architecture incorporating a combination of multiplication methods can prove more beneficial than the use of a single multiplication method for large integer multiplication. In order to test the best approach for the various multiplication sizes required in FHE schemes, the NTT, Karatsuba and Comba multiplication methods will be compared against a direct multiplication, that is the ISE instantiated multiplier unit using the available FPGA DSP48E1 blocks.

As discussed previously, each of the multiplication methods has advantages and disadvantages. For example, NTT is known to be suitable for very large integers; however, the scaling of NTT multiplication on hardware platforms is difficult. Karatsuba is faster than schoolbook multiplication, yet it requires the intermediate storage of values. In fact, the memory requirement can significantly affect performance of multiplication algorithms. In this research, as the target platform is an FPGA, all resources on the device, including memory, are limited. The use of Comba multiplication addresses this memory issue, in that it optimises the ordering of the generation of partial products and hence minimises read and write operations.

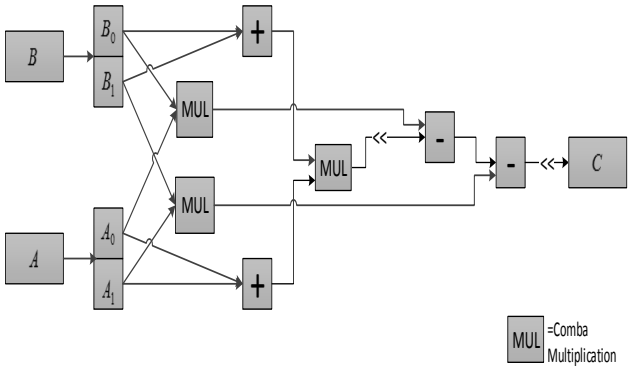

Fig. 4. Architecture of a Karatsuba multiplier with Comba multiplier units in the MUL blocks

4.1 Karatsuba-Comba

As can be noted from Equation 1 and Equation 2, a Karatsuba design employs smaller multiplication blocks, which can be interchanged. These can be seen in Figure 4 where a combination architecture using Karatsuba and Comba (Karatsuba-Comba) is given; the MUL units can use the Comba architecture. Although this may require more memory than a direct multiplication, this method maintains a reasonable clock frequency, unlike when a direct multiplication is instantiated, especially for larger multiplication sizes.

A Karatsuba-Comba combination has previously been shown to be suitable with Montgomery reduction for modular exponentiation [50]. Moreover, a software combination of Karatsuba and Comba has been previously proposed [51].

4.2 NTT Combinations

There has been an abundance of research carried out on FFT and NTT multipliers and there are numerous optimisations and moduli choices that can be selected to improve their performance, particularly for the case study of FHE [4]–[13]. However, there is limited research into hardware architectures of general purpose NTT multiplication for very large integers. In this research, a basic NTT multiplier is presented that has been optimised for area usage in order to fit the design on the target FPGA device. It employs a modulus of the form $2^{2^n}+1$, which is a Fermat number.

Akin to the Karatsuba design, an NTT design reduces a large multiplication to a series of smaller multiplications, which can be interchanged. In this research we consider several combinations.

4.2.1 NTT-Direct

The initial NTT design employs the NTT unit introduced in Section 3.3 with a direct multiplication using the FPGA DSP slices. This design is presented for comparison purposes and results are given in Section 6.4.

4.2.2 NTT-Comba

The NTT unit introduced in Section 3.3 can also be combined with the Comba multiplier instead of direct multiplication to carry out the smaller multiplications. The combined NTT and Comba design works well together in comparison to NTT-Direct as a high clock frequency can be achieved, since the multiplication unit is the bottleneck in the design.

4.2.3 NTT-Karatsuba-Comba

A multiplication architecture combining NTT, Karatsuba and Comba is also proposed. The Karatsuba multiplier in this case requires a smaller multiplication unit within its design and this can been employed with two options: direct multiplication or Comba multiplication. As Karatsuba and Comba work well together, this method is chosen and presented in the results section.

5 Hardware Complexity of Multiplication

In this section, the hardware complexity of the proposed multiplier combination designs is considered. This complexity analysis provides a generic insight into the most suitable multiplication method with respect to the operand bit length. Hardware complexity of multiplication and exponentiation has previously been considered by [17]; a similar approach is taken in this research to analyse the hardware complexity of the previously presented multipliers.

The approach for calculating the hardware complexity is defined as follows: each multiplication algorithm is recalled and the algorithms are analysed in terms of the composition of smaller units, such as gates, multiplexors and adders. In particular, the hardware complexity of the various multiplication methods are described in terms of the number of adders, and the notation $h_{add(w)}, h_{sub}(w)$ and h_{mul} is used to describe a w-bit addition, subtraction and multiplication respectively. Summations of these smaller units are used to form an expression of the hardware complexity of each multiplication method. Thus, routing and other implementation specific details are not taken into account in this analysis. Also, shifting by powers of two is considered a free operation. Four multiplication methods, that is, schoolbook, Comba, Karatsuba and NTT multiplication, are considered in the following subsections.

5.1 Complexity of Schoolbook Multiplication

Recalling the traditional schoolbook multiplication, defined in Algorithm 1, it can be seen that, for an n-bit multiplication, at most $n-1$ shifts and $n-1$ additions are required. The maximum bit length of the additions required in the schoolbook multiplication is $2n$. Thus, the hardware complexity, $h_{schoolbook}$, can be described as in Equation 5.

$$h_{schoolbook} = (n-1)h_{add}(2n) \tag{5}$$

5.2 Complexity of Comba Multiplication

The Comba multiplication algorithm is defined in Algorithm 2. This algorithm is similar to traditional schoolbook multiplication, in that the same number of operations are required and the computational complexity is the same, $O(n^2)$. However, the optimised ordering of the partial product generation improves performance, especially when embedded multipliers on the FPGA platform are targeted.

Small multiplications are required, which can be assumed to be carried out using traditional schoolbook shift and add multiplication.

The hardware complexity of the Comba multiplication is equal to h_{Comba}, given in Equation 6, where w is the bit length of the smaller multiplication blocks to generate the partial products, and w in this research is set to equal 16 bits. This multiplication can be carried out on a DSP slice, if the FPGA platform is targeted.

$$h_{Comba} = \left\lceil \frac{n}{w} \right\rceil^2 h_{mul}(w) + \left\lceil \frac{n}{w} \right\rceil^2 h_{add}(2n) \quad (6)$$

The hardware complexity of the Comba multiplication can be rewritten in terms of h_{add}, similar to the hardware complexity for the schoolbook multiplication. In this case, for each multiplication of w bits required in the Comba multiplication, it is assumed that schoolbook multiplication is used. Thus, the hardware complexity can be redefined as Equation 7.

$$h_{Comba} = \left\lceil \frac{n}{w} \right\rceil^2 ((w-1)h_{add}(2w)) + \left\lceil \frac{n}{w} \right\rceil^2 h_{add}(2n) \quad (7)$$

5.3 Complexity of Karatsuba Multiplication

The Karatsuba multiplication method is given in Algorithm 3. In this research, it is assumed firstly that the bit lengths of a_0, a_1, b_0 and b_1 are equal and set to $\frac{n}{2}$. Secondly, it is assumed that only one level of Karatsuba is used, although Karatsuba is usually employed recursively. The hardware complexity of Karatsuba multiplication, $h_{Karatsuba}$, is defined in Equation 8.

$$\begin{aligned} h_{Karatsuba} &= 2h_{add}(\tfrac{n}{2}) + 2h_{mul}(\tfrac{n}{2}) + h_{mul}(\tfrac{n}{2}+1) \\ &\quad +2h_{sub}(n) + h_{add}(n) \end{aligned} \quad (8)$$

The hardware complexity of the Karatsuba can also be written in terms of h_{add} and h_{sub}. This is given in Equation 9, where the smaller multiplications are carried out using schoolbook multiplication. Equation 10 gives the hardware complexity of Karatsuba using the Comba method for the smaller multiplications.

$$\begin{aligned} h_{K-S} &= 2h_{add}(\tfrac{n}{2}) + (n-2)h_{add}(n) \\ &\quad +(\tfrac{n}{2}+1)h_{add}(n+2) + 2h_{sub}(n) + h_{add}(n) \end{aligned} \quad (9)$$

$$\begin{aligned} h_{K-C} &= 2h_{add}(\tfrac{n}{2}) + (2[\left\lceil \tfrac{n}{2w} \right\rceil^2 (w-1)h_{add}(2w) \\ &\quad +\left\lceil \tfrac{n}{2w} \right\rceil^2 h_{add}(n)] + \left\lceil \tfrac{\frac{n}{2}+1}{w} \right\rceil^2) \times ((w-1)h_{add}(2w)) \\ &\quad +\left\lceil \tfrac{\frac{n}{2}+1}{w} \right\rceil^2 h_{add}(n+2) + 2h_{sub}(n) + h_{add}(n) \end{aligned} \quad (10)$$

5.4 Complexity of NTT Multiplication

Recall Algorithm 4 for NTT multiplication. It can be seen that the NTT requires several shift operations, additions and also multiplications. The hardware complexity of the NTT multiplication, h_{NTT}, is given in Equation 11, where k is the NTT-point, as given in the Tables of results found in Section 6.

$$\begin{aligned} h_{NTT} &= \tfrac{k}{2} \times 2h_{add}(k) + k \times h_{mul}(k) \\ &\quad +2h_{add}(k) + k \times h_{mul}(k) \\ &= (k+2)h_{add}(k) + 2k \times h_{mul}(k) \end{aligned} \quad (11)$$

The hardware complexity of the NTT can be rewritten in terms of h_{add}; this is given in Equation 12. Equations 13, 14 and 15 describe the hardware complexity of NTT-Comba, NTT-Karatsuba-Comba and NTT-Karatsuba-Schoolbook respectively.

$$h_{NTT-S} = (k+2)h_{add}(k) + 2k(k-1)h_{add}(2k) \quad (12)$$

$$\begin{aligned} h_{NTT-C} &= (k+2)h_{add}(k) \\ &\quad +2k(\left\lceil \tfrac{k}{w} \right\rceil^2 ((w-1)h_{add}(2w)) \\ &\quad +\left\lceil \tfrac{k}{w} \right\rceil^2 h_{add}(2k)) \end{aligned} \quad (13)$$

$$\begin{aligned} h_{NTT-K-C} &= (k+2)h_{add}(k) + 2k(2h_{add}(\tfrac{k}{2}) \\ &\quad +2(\left\lceil \tfrac{k}{2w} \right\rceil^2 (w-1)h_{add}(2w) \\ &\quad +\left\lceil \tfrac{k}{2w} \right\rceil^2 h_{add}(k)) + \left\lceil \tfrac{\frac{k}{2}+1}{w} \right\rceil^2 (w-1)h_{add}(2w) \\ &\quad +\left\lceil \tfrac{\frac{k}{2}+1}{w} \right\rceil^2 h_{add}(k+2) + 2h_{sub}(k) \\ &\quad +h_{add}(k)) \end{aligned} \quad (14)$$

$$\begin{aligned} h_{NTT-K-S} &= (k+2)h_{add}(k) + 2k(2h_{add}(\tfrac{k}{2}) \\ &\quad +(k-2)h_{add}(k) + (\tfrac{k}{2}+1)h_{add}(k+2) \\ &\quad +2h_{sub}(k) + h_{add}(k)) \end{aligned} \quad (15)$$

5.5 Hardware Complexity Analysis

The results of the hardware complexity analysis for a range of operand widths are discussed in this section. The weights used to calculate these results are defined in Table 1, using a similar approach employed by David *et al.* [17]. These weightings estimate a rough gate count of a full adder, with the main purpose of allowing for fair comparison across all multiplication methods. Figure 5 shows the hardware complexity trend of all of the multipliers, with the exception of Comba and Karatsuba-Comba multipliers. These two multipliers are excluded from Figure 5 as they are much larger in comparison to the other multipliers. However, Comba and Karatsuba-Comba multiplication can be useful when FPGA devices are targeted. All of the Figures indicate how each of the various multiplication methods generally scale with an increase in multiplicand bit length. It can be

TABLE 1
Weights for Addition and Subtraction units

Unit	Weight
Add	5
Sub	6

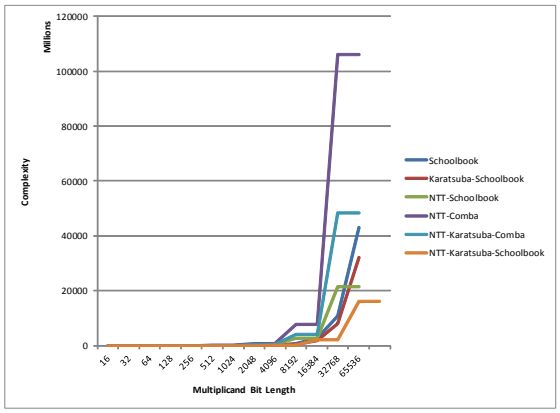

Fig. 5. Hardware complexity of multipliers

seen from Figure 5, that for much larger bit lengths, NTT-Karatsuba-Schoolbook and NTT-Schoolbook multipliers are smallest in terms of hardware complexity.

If multipliers of smaller bit lengths are considered, the suitability of various multiplication methods differs greatly. Figure 6 illustrates the most suitable multipliers for bit lengths under 512 bits. It can be seen that Karatsuba-Comba has the smallest hardware complexity for mid length operands, ranging from 64 bits to 256 bits. Karatsuba-Schoolbook is best for small operands, ranging under 64 bits. Figure 7 and Figure 8 show the hardware complexity in particular for the NTT combination multipliers. Of the NTT multipliers, and more generally for large operands, NTT-Karatsuba-Schoolbook is recommended.

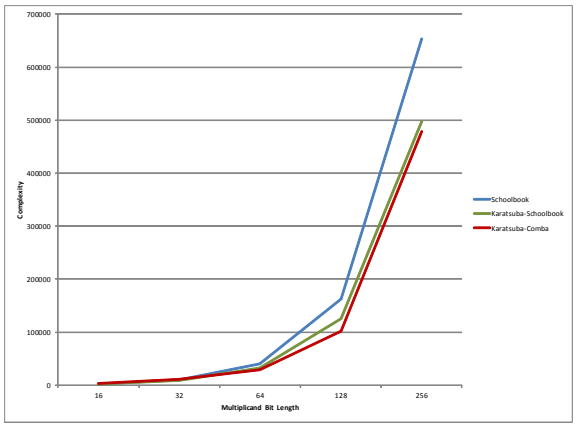

Fig. 6. Hardware complexity of a selection of multipliers for bit lengths under 512 bits

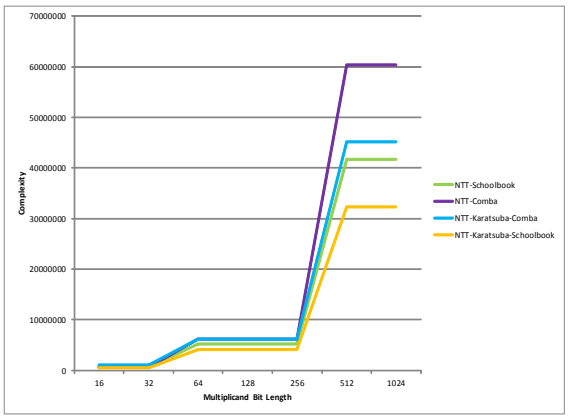

Fig. 7. Hardware complexity of a NTT combination multipliers less than 2048 bits

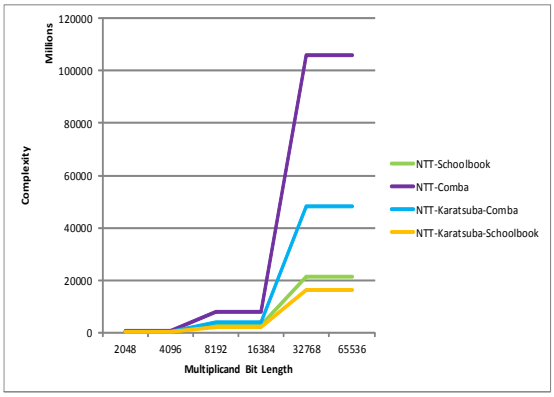

Fig. 8. Hardware complexity of a NTT combination multipliers greater than or equal to 2048 bits

6 Performance Results of Multiplier Architectures

In this section the hardware architectures proposed in Sections 3 and 4 are implemented on FPGA and the associated results are presented. A Xilinx Virtex-7 FPGA is targeted and the Xilinx ISE design suite 14.1 [52] is used throughout this research. More specifically, the target device is a Xilinx Virtex-7 XC7VX980T. This particular device is selected because it is one of the high-end Virtex-7 FPGAs [53] with a large amount of registers and the largest amount of available DSP slices (3600 DSP slices). Other FPGAs could also be considered in place of the target device. A Python script is used to generate the test vectors used in this research and a testbench is designed and used in ISE design suite to verify that the output of the multiplier unit matches the multiplication of the test vector inputs. It is to be noted that the latency results given are for a single multiplication. The multipliers can be considered as parts of larger hardware designs, and thus it is assumed that multiplication inputs are readily available on the device.

TABLE 2
Performance of direct multiplication on Virtex-7 FPGA

Bit Length	Clock Latency	Clock Frequency (MHz)	Resource Usage		
			Slice Reg	Slice LUT	DSP
16	1	491	1	1	1
32	1	493	1	1	4
64	1	518	2	159	12
128	1	490	3	1033	56
256	1	492	6	4402	231
512*	1	33.482	3150	16780	798

6.1 Direct Multiplication on FPGA

FPGAs are often specifically optimised for fast embedded multiplications, such as the Xilinx Virtex-7 FPGAs which contain embedded DSP48E1 slices. The multiplication units offered on the DSP48E1 slices have been heavily optimised to work at a high clock frequency and therefore usually offer very good performance. However, this performance gain is reduced significantly as the bit length of the required multiplication increases. In this research, the various optimised hardware multiplication designs are compared against a direct multiplication, which is an ISE instantiated multiplier unit that uses the DSP48E1 blocks on the FPGA to multiply in a single clock cycle.

Table 2 shows the hardware resource usage requirements of a direct multiplication with various bit lengths on a Virtex-7 FPGA xc7vx1140t. The asterisk, $*$, in Table 2 indicates the design IO pins are overloaded; it must be noted that this is managed using a wrapper, which incurs additional area cost. Although there are some further optimisations that can be made to improve the efficiency and the scaling of the direct multiplication, the results show the limitations of using direct multiplication and the need for alternative hardware designs specifically for large integer multiplication. This is particularly important for the area of FHE, where million-bit multiplications are required.

As can be seen from Table 2, the hardware resource usage increases rapidly. For example, if the number of required DSP48E1 slices is considered, the usage increases greatly with an increase in bit length of the multiplication operands. This trend is illustrated in Figure 9. Therefore, the use of direct multiplication is best when only smaller multiplications are required; thus it is recommended that alternative multiplication methods which scale more efficiently are considered for large multiplications such as that required in FHE schemes.

The following subsections discuss the alternatives to the direct multiplication instantiation on FPGA. These designs also target a Xilinx Virtex-7 FPGA; however they could also be used on other platforms. The results of the combinations of multipliers are also discussed within the following subsections.

6.2 Hardware Design of Comba Multiplication

Table 3 shows the performance post-place and route results of the Comba multiplication unit targeting the Xilinx Virtex-7 platform. The asterisk (*) in the table indicates the cases when the input and output pins are overloaded in a straightforward implementation, and thus this is managed by using

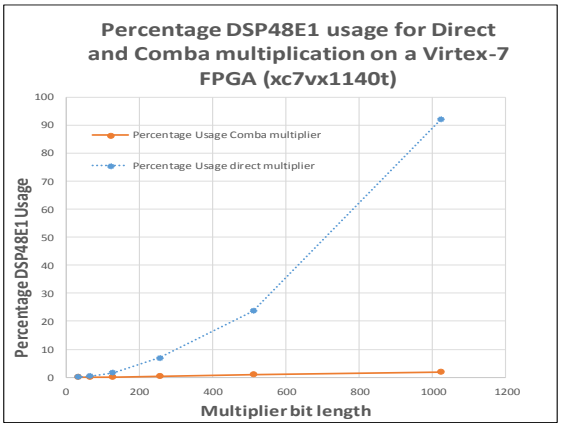

Fig. 9. Graph of the percentage usage of the DSP48E1 blocks for given direct multiplications of increasing bit-lengths on a Xilinx Virtex-7 FPGA

TABLE 3
Performance of Comba multiplication on Virtex-7 FPGA

Bit Length	Clock Latency	Clock Frequency (MHz)	Resource Usage		
			Slice Reg	Slice LUT	DSP
32	6	274.952	168	199	2
64	10	230.696	303	355	4
128	18	260.146	569	412	8
256	34	209.249	1099	907	16
512*	66	291.121	4276	4202	32
1024*	130	234.028	8439	7761	64
2048*	258	162.973	16766	15675	128
4096*	514	139.392	33407	25381	256
8192*	1026	121.507	66794	57283	512
16384*	2050	104.275	133254	141193	1024

a wrapper in the design, which incurs additional resources to store the input and output registers. As can be seen in Table 3, the number of DSP slices required increases slowly with an increase in multiplication operand bit length, unlike in Table 2 for the direct multiplication unit. These trends can be seen clearly in Figure 9; less than two percent of the available DSP resources are used for a 1024-bit Comba multiplier. Additionally, although in general more resources are initially required for the Comba multiplication unit for smaller operands, the usage scales slowly with the increase in bit length.

6.3 Hardware Design of Karatsuba-based Multiplication

The Karatsuba multiplier design using direct multiplication (Karatsuba-Direct) and also the Karatsuba-Comba multiplier design have both been implemented on a Xilinx Virtex-7 FPGA. Table 4 gives the performance results of the Karatsuba-Direct multiplier. The Karatsuba-Direct multiplication approach results in a slower clock frequency, due to the scaling limitations associated with the direct multiplication that have been previously mentioned. Therefore, the Karatsuba-Comba multiplier performs better. Table 5 shows the post-place and route performance results of the Karatsuba-Comba multiplier. This design uses more

TABLE 4
Performance of Karatsuba-Direct multiplication on Virtex-7 FPGA

Bit Length	Clock Latency	Clock Frequency (MHz)	Resource Usage		
			Slice Reg	Slice LUT	DSP
128	31	104.318	3687	3499	38
256*	49	74.512	8214	7452	181
512*	85	51.698	15909	20474	686

TABLE 5
Performance of Karatsuba-Comba multiplication on Virtex-7 FPGA

Bit Length	Clock Latency	Clock Frequency (MHz)	Resource Usage		
			Slice Reg	Slice LUT	DSP
128	42	351.617	4049	3266	13
256	49	291.630	7657	7605	25
512*	65	236.855	17210	11219	49
1024*	97	160.720	34083	22995	97
2048*	161	96.237	67816	36236	193
4096*	289	50.075	135279	82122	385

hardware resources but has a lower latency than solely the Comba multiplier design, as can be seen if Table 3 and Table 5 are compared. The latency is impacted greatly with the choice of adder. The latency of the adder depends solely on the *add width*, denoted as a_w, that is the width of the smaller blocks which are sub-blocks of the input blocks to be added. Thus, a trade-off exists, such that the use of a larger a_w decreases the latency but also decreases the achievable clock frequency of the design. In this design, a_w is set to equal one quarter of the multiplicand bit length, allowing the adder block to increase in size with an increase in bit length. This minimises the latency but for larger multiplicand bit lengths this choice of a_w significantly limits the achievable clock frequency. Therefore, a_w should be adjusted appropriately depending on the target multiplicand bit length.

The hardware design proposed in this research for Karatsuba multiplication is optimised but does not use the Karatsuba algorithm recursively; this design decision is made to minimise the use of hardware resources on the FPGA platform, especially for larger multiplications. Thus, it should be noted that Karatsuba is a fast multiplication method, and an improved hardware design of the Karatsuba algorithm, which uses the algorithm recursively without incurring too much hardware resource cost could offer better performance gains. As mentioned in Section 4.1, Karatsuba and Comba have been combined on software and showed promise. Although there have been several proposed software designs of Karatsuba and Comba and also combined with Montgomery multiplication, no hardware designs of Karatsuba and Comba multiplication can currently be found in the literature. Therefore, this is one of the first proposed hardware designs of Karatsuba and Comba.

6.4 Hardware Design of NTT Multiplication

Table 6 and Table 7 give the hardware resource usage and the clock latency of the NTT-Direct and the NTT-Comba multiplication designs respectively. These tables show that a large amount of area resources are required. Highly optimised NTT hardware designs are required for FHE schemes to minimise resource usage.

TABLE 6
Performance of NTT-Direct multiplication on Virtex-7 FPGA

Bit Length	NTT Point	Clock Latency	Clock Frequency (MHz)	Resource Usage		
				Slice Reg	Slice LUT	DSP
64*	64	3750	105.407	61021	73853	6
128*	64	3900	104.526	62184	74163	6
256*	64	4150	104.712	64489	75020	6

TABLE 7
Performance of NTT-Comba multiplication on Virtex-7 FPGA

Bit Length	NTT Point	Clock Latency	Clock Frequency (MHz)	Resource Usage		
				Slice Reg	Slice LUT	DSP
64*	64	4400	114.732	61273	73940	4
128*	64	4500	118.779	62433	74694	4
256*	64	4750	118.161	64745	75366	4

Similarly to the Karatsuba multiplier, the NTT multiplication unit has an increased clock frequency when combined with the Comba multiplication unit. The hardware resource usage could be further reduced and the clock frequency further increased through the deployment of several known optimisations.

Table 8 gives the post-place and route hardware resource usage and clock latency of the NTT-Karatsuba-Comba multiplier. In Table 6, Table 7 and Table 8 the clock latency values are rounded up to the nearest fifty as the latency can vary slightly between multiplications. It can be seen in Table 8 that the combination of NTT-Karatsuba-Comba leads to a larger design with more latency and therefore currently offers no advantages over the NTT-Comba multiplier. This result shows that some combination multipliers can lead to increased overhead. The hardware combination of multipliers should therefore be carefully considered with respect to the target application.

It can be seen in Table 7, that the resource usage increases greatly with an increase in the NTT point, i.e. when the bit length increases over a given threshold. This is because the number of required butterflies in each stage and the number of stages in an NTT architecture is dependent on the NTT point.

The NTT hardware multiplier design in this research could also be further improved. Two multiplication units and two NTT units could be used to reduce latency. In addition to this, the butterfly units could be serially implemented instead of a parallel implementation for each stage of the NTT. These optimisations would improve the performance.

TABLE 8
Performance of NTT-Karatsuba-Comba multiplication on Virtex-7 FPGA

Bit Length	NTT Point	Clock Latency	Clock Frequency (MHz)	Resource Usage		
				Slice Reg	Slice LUT	DSP
64*	64	8300	110.8033	63017	73250	7
128*	64	8450	110.156	64179	73713	7
256*	64	8700	102.354	66485	75499	7

However, all optimisations have a trade-off in terms of either increased latency or increased hardware resources and thus the design optimisations depend greatly on the motivation of the design. Moreover, it must be mentioned that, although the NTT multiplier architecture has a large latency, due to the inherent and regular structure of the NTT this architecture can be suitably pipelined to achieve a high throughput. This is advantageous in applications which require several multiplication operations.

6.5 Clock Cycle Latency and Clock Frequency for Multipliers

The clock cycle latency required for the different multipliers is given in Table 9 for comparison purposes. This clearly shows that the NTT design used in this research does require considerably more clock cycles for a multiplication when compared to the other methods. Moreover, the Karatsuba-Comba design presented in this research offers a reduced latency for multiplicand bit lengths greater than 512 bits compared to Comba multiplication.

Table 9 also compares the clock frequencies to give an idea of which multiplier operates the fastest. As can be seen in the table, the Comba multiplication has the highest clock frequency. Additionally, it must be noted that the clock frequency of both the NTT and the Karatsuba designs improve when combined with the Comba multiplication unit. The clock frequency of the Karatsuba-Comba design decreases rapidly with increased bit length; as previously mentioned, the adder used within the Karatsuba-Comba design has an impact on this clock frequency. Therefore, this research shows that there are potential benefits in using combined multiplier architectures, depending on the application specifications.

The latency of each of the multipliers can be described more generically, to give estimates for any multiplication bit length. The latency for the Direct multiplier is equal to 1 clock cycle for any multiplication bit length. Let w be the multiplication width and s be the small multiplication block width, used within the DSP slices (s is set to 16-bits in the Comba design). Then, the latency of the Comba multiplier is given in Equation 16. As the Comba design employs the DSP slices to calculate the partial products required in large multiplication in a scheduled manner, the latency is directly associated with the number of required DSP slices, which is $\frac{w}{s}$, and there is also a small overhead for partial product accumulation.

$$2\left\lceil \frac{w}{s} \right\rceil + 2 \tag{16}$$

The latency of the Karatsuba designs also depend on w and s and additionally a, the addition block width. The latency of the Karatsuba designs is calculated by summing the latency of one small multiplier, the adders and an additional constant latency requirement of 4 clock cycles. One $\frac{w}{2} + s$-bit multiplier is required. Also, four additions are required, which are of the size $\frac{w}{2}$-bit, $w + 2s$-bit, $w + 3s$-bit and $2w + 1$-bit respectively. The latency of each adder is set to $\lceil \frac{w_{add}}{a} \rceil + 1$, where w_{add} is the maximum bit length of the elements to be added. The latency of Karatsuba-Direct is given in Equation 17. Within the Karatsuba-Direct design, the addition block width is set to equal $a = 32$. Within the Karatsuba-Comba design, the addition block width is set, such that $a = \frac{w}{4}$. Equation 18 gives the latency for the Karatsuba-Comba design. For any values greater than 192-bits, Equation 18 is equivalent to Equation 19.

$$\left\lceil \frac{w}{2a} \right\rceil + 2\left\lceil \frac{w}{a} \right\rceil + \left\lceil \frac{5s}{a} \right\rceil + \left\lceil \frac{2w+1}{a} \right\rceil + 9 \tag{17}$$

$$\left\lceil \frac{w}{s} \right\rceil + \left\lceil \frac{8s}{w} \right\rceil + \left\lceil \frac{12s}{w} \right\rceil + \left\lceil \frac{4}{w} \right\rceil + 30 \tag{18}$$

$$\left\lceil \frac{w}{s} \right\rceil + 33 \tag{19}$$

Lastly, an estimation for the latency of the NTT combination architectures is given in Equation 20, where w' is the NTT point size and m is the latency of the multiplication, either Comba, Direct or Karatsuba, as defined above. Also, r is the latency of the modular reduction step, b is the latency of the NTT butterflies and t is the latency of the addition step. In the modular reduction step, a maximum of 2 additions are required as well as an additional 2 clock cycles. As the addition block width is set to equal the width of the entire addition, the addition requires 2 clock cycles. Thus, in this case $r = 4$. The latency of the butterfly operations is estimated in this case as $b = 21$ and the latency of the addition step is estimated as $t = 4$.

$$3b\,log_2(w') + 2(w'm + w'r) + \frac{w'(w'-1)}{2} + (w'-1)t \tag{20}$$

A graph is given in Figure 10 that compares the latencies of all of the multiplier methods, with the exception of the NTT combinations, as these require much greater latencies. This graph highlights the impact the multiplication bit length has on the performance of these designs. It can be seen that for larger numbers Karatsuba-Direct has the highest latency and Karatsuba-Comba has the lowest latency, not including the Direct multiplication, which has a low latency but requires much greater area resources with each increase in multiplication bit length and thus is not a feasible option for large integers.

7 Comparison for Uneven Operands

An alternative approach to a regular square multiplier is sometimes required for various applications, for example in the case of the encryption step in the FHE scheme over the integers, see [23]. The multiplicands within the large integer multiplication step differ greatly in size. In order to investigate this further, the multiplication methods presented in this research are also employed within an uneven multiplication unit. This unit is depicted in Figure 11. In this design, the MUL unit is interchanged to measure the performance of various multiplication methods.

For the case of integer based FHE, as proposed by [23], a much smaller multiplicand, b_i, is required in the encryption step, which ranges from 936 bits to 2556 bits. Thus, a

TABLE 9
Clock cycle latency and frequency (MHz) of multipliers with respect to bit length

Bit Length	Comba		Karatsuba + Direct		Karatsuba + Comba		NTT + Direct		NTT + Comba	
	Latency	clock freq.	Latency	clock freq.	Latency	clock freq.	Latency	clock freq.	Latency	clock freq.
64	10	230.696	-	-	-	-	3750	105.407	4400	114.732
128	18	260.146	31	104.318	42	351.617	3900	104.526	4500	118.779
256	34	209.249	49	74.512	49	291.630	4150	104.712	4750	118.161
512	66	291.121	85	51.698	65	236.855	-	-	-	-
1024	130	234.028	-	-	97	160.720	-	-	-	-
2048	258	162.973	-	-	161	96.237	-	-	-	-
4096	514	139.392	-	-	289	50.075	-	-	-	-
8192	1026	121.507	-	-	-	-	-	-	-	-
16384	2050	104.275	-	-	-	-	-	-	-	-

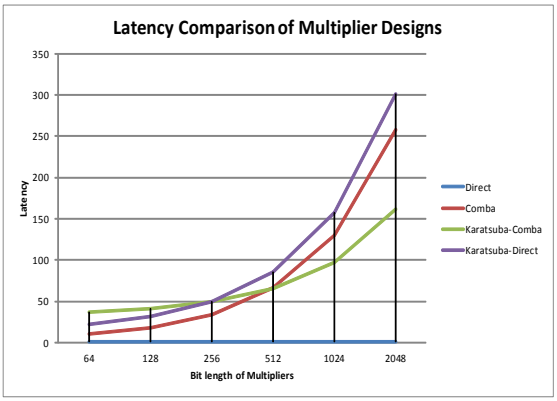

Fig. 10. Latency of four of the multipliers

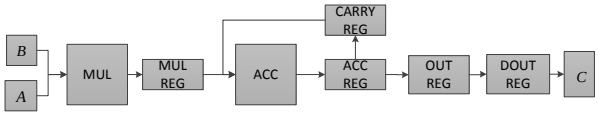

Fig. 11. Architecture of the uneven multiplication unit

smaller square multiplication unit, of the size of the smaller multiplicand, is reused in this design for the multiplication with uneven operands and the subsequent partial products are accumulated to produce the final output.

Table 10 presents latency results for the uneven multiplication unit with respect to several multiplication methods. A selection of bit lengths are investigated. There are several assumptions in this design that must be taken into consideration when analysing the results. Firstly, the multiplication methods are not pipelined as only one multiplication is considered here for comparison purposes. Of the current designs presented in this research, Table 10 shows that Comba is most suitable for uneven operands, due to the relatively high clock frequency and low latency achievable.

TABLE 10
Clock cycle latency and hardware resource usage of multipliers within an uneven multiplication

Multiplier Type	Latency	Clock Freq (MHz)	Slice Reg	Slice LUT	DSP
Bit length of A = 64; Bit length of B = 128					
Comba	56	399.457	1401	1077	4
Karatsuba-Comba	136	399.457	3417	2781	7
Direct	26	109.077	1099	753	12
Bit length of A = 256; Bit length of B = 1024					
Comba	226	212.981	2182	1587	16
Karatsuba-Comba	316	84.328	7400	8105	181
NTT-Comba	25550	127.159	86436	99123	18
Direct	38	63.930	2074	5679	231
Bit length of A = 512; Bit length of B = 1024					
Comba	280	122.680	10871	6870	32
Karatsuba-Comba	496	122.680	24240	13004	49
NTT-Comba	16300	122.680	273527	405246	8
Direct	26	42.405	8737	18921	798
Bit length of A = 1024; Bit length of B = 4096					
Comba	802	66.386	27833	11220	64
Karatsuba-Comba	1372	66.386	51492	27408	97
Bit length of A = 1024; Bit length of B = 8192					
Comba	1334	66.386	27833	11220	64
Karatsuba-Comba	2284	66.386	63781	29454	97
Bit length of A = 1024; Bit length of B = 16384					
Comba	2398	66.386	64699	20888	64
Karatsuba-Comba	4108	66.386	88358	33553	97

8 Conclusions

In this paper, the hardware designs of several large integer multipliers were proposed and a hardware complexity analysis was also given for each of the most common multiplication methods. In conclusion, the hardware results of the proposed multiplier combination designs show that Karatsuba-Comba offers low latency at the cost of additional area resources in comparison to a hardware Comba multiplier. Additionally, Comba is shown to be the most suitable multiplication method when uneven operand multiplication is required.

Moreover, it can be seen from the hardware complexity analysis and the latency analysis, that the bit length range of the operands is an important factor in the selection of a suitable multiplication method. The hardware complexity figures give an idea of how these combination multipliers will generally scale, without targeting any specific platform. Generally, the results of the hardware complexity analysis

show that NTT-Karatsuba-Schoolbook multiplication is the best choice for very large integers. Other factors must also be considered when selecting multipliers, such as the optimisation target, for example low area or high speed. Another factor is the algorithm to be implemented and the associated computations required for such algorithms other than multiplication, which will potentially dictate the amount of available resources on the target device for multiplication and thresholds on latency for multipliers within the entire implementation.

REFERENCES

[1] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, 2009, URL: http://crypto.stanford.edu/craig.

[2] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, 2012, pp. 1219–1234.

[3] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "Cloud-assisted multiparty computation from fully homomorphic encryption," IACR Cryptology ePrint Archive, Report 2011/663, 2011.

[4] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," Cryptology ePrint Archive, Report 2011/566, 2011.

[5] C. Gentry, S. Halevi, C. Peikert, and N. P. Smart, "Ring switching in BGV-style homomorphic encryption," in *Security and Cryptography for Networks - 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings*, 2012, pp. 19–37.

[6] S. Halevi and V. Shoup. (2012) HElib, homomorphic encryption library. https://github.com/shaih/HElib.

[7] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings*, 2012, pp. 139–158.

[8] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Accelerating fully homomorphic encryption using GPU," in *IEEE Conference on High Performance Extreme Computing, HPEC 2012, Waltham, MA, USA, September 10-12, 2012*, 2012, pp. 1–5.

[9] W. Wang and X. Huang, "FPGA implementation of a large-number multiplier for fully homomorphic encryption," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013), Beijing, China, May 19-23, 2013*, 2013, pp. 2589–2592.

[10] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Exploring the feasibility of fully homomorphic encryption," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, p. 1, 2013.

[11] W. Wang, X. Huang, N. Emmart, and C. C. Weems, "VLSI design of a large-number multiplier for fully homomorphic encryption," *IEEE Trans. VLSI Syst.*, vol. 22, no. 9, pp. 1879–1887, 2014.

[12] X. Cao, C. Moore, M. O'Neill, N. Hanley, and E. O'Sullivan, "High speed fully homomorphic encryption over the integers," in *Financial Cryptography and Data Security - FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers*, 2014, pp. 169–180.

[13] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, and N. Hanley, "Optimised multiplication architectures for accelerating fully homomorphic encryption," *IEEE Trans. Computers*, vol. 65, no. 9, pp. 2794–2806, 2016. [Online]. Available: http://dx.doi.org/10.1109/TC.2015.2498606

[14] Y. Doröz, E. Öztürk, and B. Sunar, "Evaluating the hardware performance of a million-bit multiplier," in *16th Euromicro Conference on Digital System Design (DSD)*, 2013, pp. 955–962.

[15] Y. Doröz, E. Öztürk, and B. Sunar, "A million-bit multiplier architecture for fully homomorphic encryption," *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 38, no. 8, pp. 766–775, 2014.

[16] C. Moore, M. O'Neill, N. Hanley, and E. O'Sullivan, "Accelerating integer-based fully homomorphic encryption using Comba multiplication," in *2014 IEEE Workshop on Signal Processing Systems, SiPS 2014, Belfast, United Kingdom, October 20-22, 2014*, 2014, pp. 62–67.

[17] J. David, K. Kalach, and N. Tittley, "Hardware complexity of modular multiplication and exponentiation," *IEEE Trans. Computers*, vol. 56, no. 10, pp. 1308–1319, 2007.

[18] I. San and N. At, "On increasing the computational efficiency of long integer multiplication on FPGA," in *11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2012, Liverpool, United Kingdom, June 25-27, 2012*, 2012, pp. 1149–1154.

[19] A. Abdel-Fattah, A. Bahaa El-Din, and H. Fahmy, "Modular multiplication for public key cryptography on FPGAs," in *Computer Sciences and Convergence Information Technology, 2009. ICCIT '09. Fourth International Conference on*, Nov 2009, pp. 1096–1100.

[20] C. McIvor, M. McLoone, and J. McCanny, "Fast Montgomery modular multiplication and RSA cryptographic processor architectures," in *37th Asilomar Conference on Signals, Systems and Computers*, 2003, pp. 379–384.

[21] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster interleaved modular multiplication based on Barrett and Montgomery reduction methods," *IEEE Trans. Computers*, vol. 59, no. 12, pp. 1715–1721, 2010.

[22] S. Srinivasan and A. Ajay, "Comparative study and analysis of area and power parameters for hardware multipliers," in *Electrical, Electronics, Signals, Communication and Optimization (EESCO), 2015 International Conference on*, Jan 2015, pp. 1–5.

[23] J.-S. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic encryption over the integers," in *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, 2012, pp. 446–464.

[24] P. G. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM Systems Journal*, vol. 29, no. 4, pp. 526–538, 1990.

[25] L. Malina and J. Hajny, "Accelerated modular arithmetic for low-performance devices," in *34th International Conference on Telecommunications and Signal Processing (TSP 2011), Budapest, Hungary, Aug. 18-20, 2011*, 2011, pp. 131–135.

[26] T. Güneysu, "Utilizing hard cores of modern FPGA devices for high-performance cryptography," *J. Cryptographic Engineering*, vol. 1, no. 1, pp. 37–55, 2011.

[27] A. A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics Doklady*, vol. 7, pp. 595–596, 1963, URL: `http://cr.yp.to/bib/entries.html#1963/karatsuba`.

[28] G. C. T. Chow, K. Eguro, W. Luk, and P. H. W. Leong, "A Karatsuba-based Montgomery multiplier," in *International Conference on Field Programmable Logic and Applications, FPL 2010, August 31 2010 - September 2, 2010, Milano, Italy*, 2010, pp. 434–437.

[29] N. Nedjah and L. de Macedo Mourelle, "A review of modular multiplication methods and respective hardware implementation," *Informatica (Slovenia)*, vol. 30, no. 1, pp. 111–129, 2006.

[30] P. L. Montgomery, "Five, six, and seven-term Karatsuba-like formulae," *IEEE Trans. Computers*, vol. 54, no. 3, pp. 362–369, 2005.

[31] C. C. Corona, E. F. Moreno, and F. Rodríguez-Henríquez, "Hardware design of a 256-bit prime field multiplier suitable for computing bilinear pairings," in *2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011, Cancun, Mexico, November 30 - December 2, 2011*, 2011, pp. 229–234.

[32] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," *Soviet Mathematics Doklady*, vol. 3, pp. 714–716, 1963.

[33] S. A. Cook, "On the minimum computation time of functions," Ph.D. dissertation, 1966, URL: `http://cr.yp.to/bib/entries.html#1966/cook`.

[34] GMP, "GMP library: Multiplication," 2014, URL: https://gmplib.org/manual/Multiplication-Algorithms.html.

[35] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[36] A. Schönhage and V. Strassen, "Schnelle Multiplikation großer Zahlen," *Computing*, vol. 7, no. 3-4, pp. 281–292, 1971.

[37] N. Emmart and C. C. Weems, "High precision integer multiplication with a GPU using Strassen's algorithm with multiple FFT sizes," *Parallel Processing Letters*, vol. 21, no. 3, pp. 359–375, 2011.

[38] L. Ducas and D. Micciancio. (2014) A fully homomorphic encryption library. https://github.com/lducas/FHEW.

[39] J. A. Solinas, "Generalized Mersenne numbers," 1999, tech Report.

[40] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. Cheung, D. Pao, and I. Verbauwhede, "High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems," Cryptology ePrint Archive, Report 2014/646, 2014.

[41] T. Gyorfi, O. Cret, G. Hanrot, and N. Brisebarre, "High-throughput hardware architecture for the SWIFFT / SWIFFTX hash functions," Cryptology ePrint Archive, Report 2012/343, 2012.
[42] L. M. Leibowitz, "A simplified binary arithmetic for the fermat number transform," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 24, no. 5, pp. 356–359, 1976.
[43] K. Kalach and J. P. David, "Hardware implementation of large number multiplication by FFT with modular arithmetic," *3rd International IEEE-NEWCAS Conference*, pp. 267–270, 2005.
[44] C. Cheng and K. K. Parhi, "High-throughput VLSI architecture for FFT computation," *IEEE Trans. on Circuits and Systems*, vol. 54-II, no. 10, pp. 863–867, 2007.
[45] S. Baktir and B. Sunar, "Achieving efficient polynomial multiplication in fermat fields using the fast Fourier transform," in *Proceedings of the 44st Annual Southeast Regional Conference, 2006, Melbourne, Florida, USA, March 10-12, 2006*, 2006, pp. 549–554.
[46] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-LWE cryptoprocessor," in *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, 2014, pp. 371–391.
[47] C. Moore, N. Hanley, J. McAllister, M. O'Neill, E. O'Sullivan, and X. Cao, "Targeting FPGA DSP slices for a large integer multiplier for integer based FHE," in *Financial Cryptography and Data Security - FC 2013 Workshops, USEC and WAHC 2013, Okinawa, Japan, April 1, 2013, Revised Selected Papers*, 2013, pp. 226–237.
[48] Xilinx. (2013) 7 series DSP48E1 Slice. [Online]. Available: http://www.xilinx.com/ support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
[49] Xilinx. (2014) 7 series FPGAs overview. [Online]. Available: http://www.xilinx.com/ support/documentation/data_sheets/ds180_7Series_Overview.pdf
[50] M. P. Scott, "Comparison of methods for modular exponentiation on 32-bit Intel 80x86 processors," 1996. [Online]. Available: goo.gl/SxGkGD
[51] J. Großschädl, R. M. Avanzi, E. Savas, and S. Tillich, "Energy-efficient software implementation of long integer modular arithmetic," in *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, 2005, pp. 75–90.
[52] Xilinx. (2015) ISE design suite. [Online]. Available: http://www.xilinx.com/ products/design-tools/ise-design-suite.html
[53] Xilinx. (2014) 7 series FPGAs overview. [Online]. Available: http://www.xilinx.com/ support/documentation/data_sheets/ds180_7Series_Overview.pdf

Ciara Rafferty (M'14) received first-class honours in the BSc. degree in Mathematics with Extended Studies in Germany at Queen's University Belfast in 2011 and the Ph.D. degree in electrical and electronic engineering from Queen's University Belfast in 2015. She is currently a Research Assistant in Queen's University Belfast. Her research interests include hardware cryptographic designs for homomorphic encryption and lattice-based cryptography.

Máire O'Neill (M'03-SM'11) received the M.Eng. degree with distinction and the Ph.D. degree in electrical and electronic engineering from Queen's University Belfast, Belfast, U.K., in 1999 and 2002, respectively. She is currently a Chair of Information Security at Queen's and previously held an EPSRC Leadership fellowship from 2008 to 2015. and a UK Royal Academy of Engineering research fellowship from 2003 to 2008. She has authored two research books and has more than 115 peer-reviewed conference and journal publications. Her research interests include hardware cryptographic architectures, lightweight cryptography, side channel analysis, physical unclonable functions, post-quantum cryptography and quantum-dot cellular automata circuit design. She is an IEEE Circuits and Systems for Communications Technical committee member and was Treasurer of the Executive Committee of the IEEE UKRI Section, 2008 to 2009. She has received numerous awards for her research and in 2014 she was awarded a Royal Academy of Engineering Silver Medal, which recognises outstanding personal contribution by an early or mid-career engineer that has resulted in successful market exploitation.

Neil Hanley received first-class honours in the BEng. degree, and the Ph.D. degree in electrical and electronic Engineering from University College Cork, Cork, Ireland, in 2006 and 2014 respectively. He is currently a Research Fellow in Queen's University Belfast. His research interests include secure hardware architectures for post-quantum cryptography, physically unclonable functions and their applications, and securing embedded systems from side-channel attacks.